EV549907871

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application Serial No.: 09/417,990
Filing Date: October 13, 1999
Inventors: Lovett et al.
Applicant: Microsoft Corporation
Group Art Unit: 2178
Examiner: Queler
Confirmation No.: 8254
Attorney's Docket No.: MS1-383US
Title: Methods and Systems for Processing XML Documents

DECLARATION UNDER 37 C.F.R. §1.131

As a below named inventor, I hereby declare that:

My residence, post office address, and citizenship are as stated below next to my name. I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled "Methods and Systems for Processing XML Documents," as identified above.

The invention was conceived in the United States prior to July 1, 1999, which is prior to the publication date of "XML Authority Ends Waiting Games for Schema Developers", by Dougherty (published 7/1/1999) and XML Authority™ by Extensibility Inc. (published 8/1/1999).

Attached to this declaration is a redacted copy of a document entitled "XML Schema Patent Disclosure" that I prepared prior to July 1, 1999 and submitted to our in-house patent department during the first week of July 1999. This document evidences that the invention was conceived prior to July 1, 1999, which predates the two publications mentioned above.

I diligently pursued filing the above identified application from prior to July 1, 1999, until the filing of the above identified application on October 13, 1999, including discussing the invention with a patent attorney during a disclosure meeting and reviewing draft copies of the patent application.

All statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statement may jeopardize the validity of the application or any patent issued therefrom.

* * * * * * * *

Full name of inventor: Christopher J. Lovett

Inventor's Signature _____ Date: 1/24/05

Residence: Woodinville, WA

Citizenship: Australia

Post Office Address: 18606 201st Ave NE
Woodinville, WA 98072

Full name of inventor: Nanshan Zeng

Inventor's Signature _____ Date: 1/24/05

Residence: Bellevue, WA

Citizenship: People's Republic of China

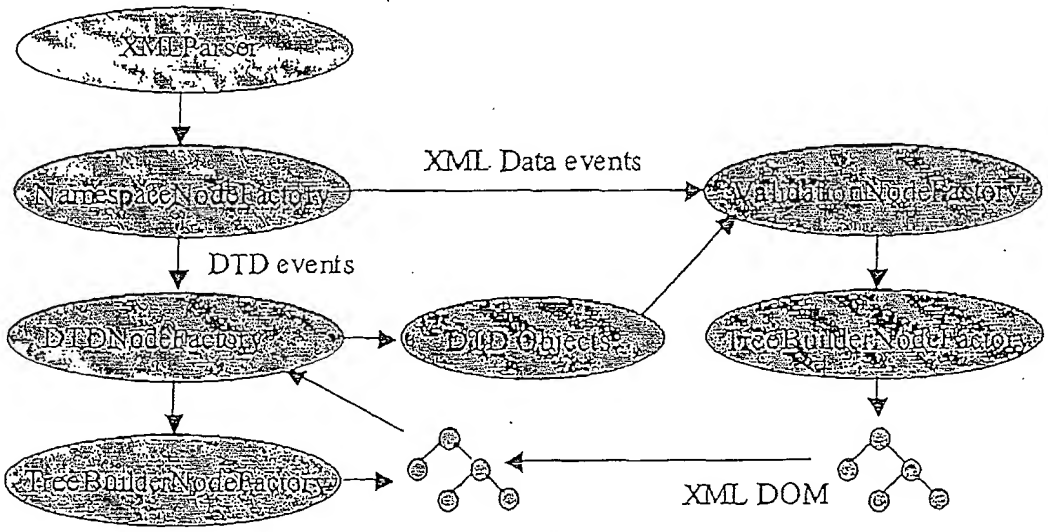
Post Office Address: 7029 169th Ave SE
Bellevue, WA 98006

XML Schema Patent Disclosure 7/7/99

Motivation for Invention or Problem Addressed

First of all there is motivation for the concept of DTD Validation in general. This comes up in the internet when one company receives XML data from some random end user. The company receiving the data doesn't really trust where the data comes from and a process of "validation" determines whether the data is good and can continue with further processing or whether it is just random noise that can be rejected outright or sent to a human for special consideration. DTD or XML-Data Schema Validation makes the subsequent processing easier because it can make simplifying assumptions about the shape and content of that data. A simple example is that a dollar amount has a valid fixed decimal format and can therefore be used successfully in subsequent calculations. There are not really any white papers on "why validation?" It is pretty much an accepted requirement for automated processing of XML documents that there be a mechanism for defining rigorous constraints on the shape and content of those documents.

The following diagram illustrates the existing node factory, and DTD world. The XMLParser calls a NamespaceNodeFactory which passes DTD events to a DTDNodeFactory. The DTDNodeFactory builds an in memory representation of the labeled "DTD Objects" and sometimes delegates to a TreeBuilderNodeFactory which builds some XML Document Object Model fragments for some pieces of the DTD. After parsing the DTD NamespaceNodeFactory then delegates to the ValidationNodeFactory which uses the DTD objects to apply the validation constraints on the xml data. The ValidationNodeFactory then also delegates to the TreeBuilderNodeFactory which builds the real XML Document Object Model for the data. Some elements in the XML DOM reference fragments in the DTD DOM (for example, entities are handled this way).



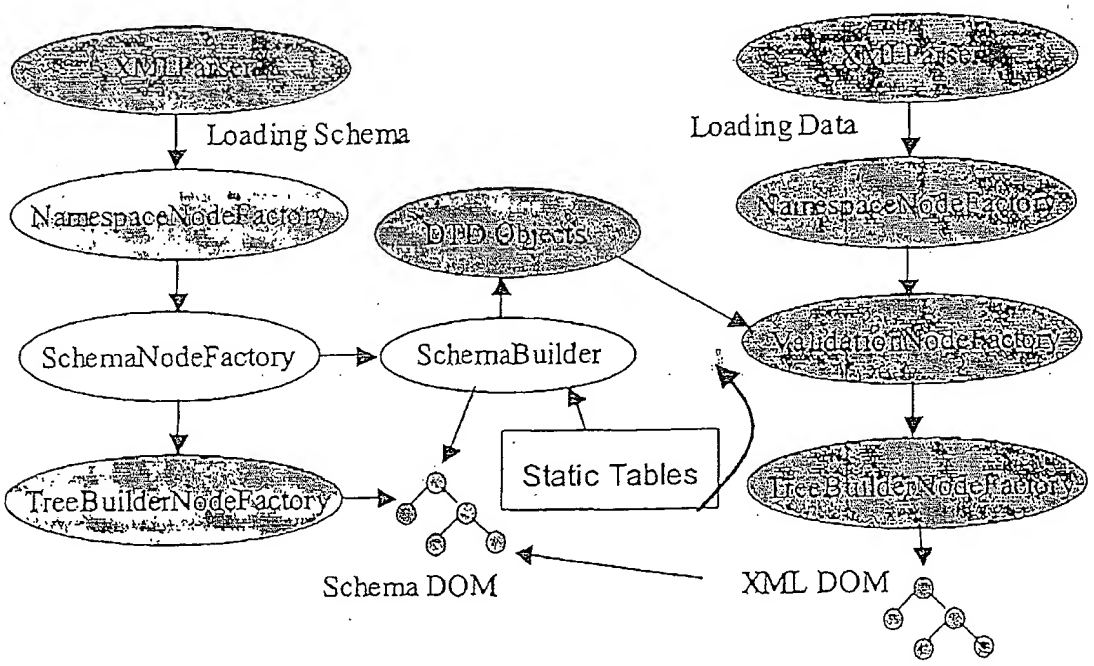
DTD DOM Fragments

The problem was to create an IXMLNodeFactory (the output of our XML Parser – see that efficiently implements that spec and that leaverages the existing implementation of DTD support required by the XML 1.0 Language Specification (http://www.w3.org/TR/PR-xml-971208)

The problem is that the IXMLNodeFactory interface provides a sequential stream of XML tokens, but the XML-Data spec defines things in a way that is order independent. This means the node factory has to store up certain state until it knows it can process that state. The node factory was implemented first by an intern Calvin Shen and was largely redesigned by Nanshan Zeng with help from Chris Lovett because of architectural problems in Calvin's version.

Description of the Invention

The solution starts with a SchemaNodeFactory which is a small IXMLNodeFactory implementation that delegates all of the work to a class called SchemaBuilder. The Schema builder is a table driven schema parser that build DTD objects. The SchemaBuilder also uses the XML in memory tree representation built by the TreeBuilder in order to extract the post-entity-expanded text values for attributes. This new stuff (shown in white) then plugs nicely into the existing NodeFactory design (shown in gray).



SchemaBuilder API

The Schema Builder Api is very simple and is designed so that it can be used in other scenarios outside of the parse time SchemaNodeFactory scenario. It consists of the following methods:

[redacted]

This method signals that a new schemas is being processed and the internal state needs to be reset accordingly.

[redacted]

This method signals that the end of the schema has been reached. This method checks to make sure all elements that have been referenced in the schema have been declared.

[redacted]

This method is called for each schema element in the document. This is when the SchemaBuilder looks up the new element in the table of allowable elements for the current schema element. If it is allowed, it pushes the current element on the stack and then calls the "init" function defined in the state table for the new schema element (e.g. ElementType or AttributeType). The init functions initializes some data structures for the new schema element and creates empty DTD objects for that element (e.g. ElementDecl or AttDef).

[redacted]

This method is called when all the attributes are ready to be processed for the current schema element. This is when the SchemaBuilder walks through the attributes in the XML DOM for the schema and looks them up in the attribute table for the current schema element, parses the attribute value according to the attribute type in the table and calls the function for handling that attribute. Then at the end it also calls a function for

[redacted]

This method is called for each Text node (including attribute values) in the XML document. This method determines whether it is valid to have text in the current position within the schema document and generates an error if it is not valid.

[redacted]

This method is called when we reach the end of all the children for the current schema element. This is when the SchemaBuilder figures out if all the content for the current schema element has been correctly provided and figures out any final default values for the current schema element. For ElementType elements it also finishes up the content model which may involve creating a few more DTD ContentModel nodes.

Example

The following table shows an example of how the CreateNode calls from the parser are translated into DTD object through the above process. Taking as input the following schema fragment:

[redacted]

SchemaNodeFactory call	SchemaBuilder call	Resulting action
NotifyEvent STARTDOCUMENT	Start	Init state and create new empty DTD obejct
CreateNode ELEMENT "Schema"	ProcessElementNode	set current state to point to the Schema element state table

CreateNode ATTRIBUTE "xmlns"	Noop	
CreateNode TEXT "urn:..."	ProcessPCDATA	Attribute text is a noop
BeginChildren	ProcessAttributes	"xmlns" is a skippable attribute so this is a noop
CreateNode ELEMENT "ElementType"	ProcessElementNode	"ElementType" is an allowable child of "Schema" so push "Schema" onto the stack and initialize a new ElementDecl DTD object with default content model EMPTY.
CreateNode ATTRIBUTE "name"	noop	
CreateNode TEXT "foo"	ProcessPCDATA	Attribute text is a noop
CreateNode ATTRIBUTE "model"	noop	
CreateNode TEXT "open"	ProcessPCDATA	Attribute text is a noop
BeginChildren	ProcessAttributes which finds the attributes and looks them up finds that they are valid and calls buildElementName and buildElementModel accordingly.	buildElementName initializes the name of the ElementDecl object. It also validates that an ElementDecl named "foo" hasn't already been declared. buildElementModel validates the value "open" and sets the model to open.
EndChildren "ElementType"	ProcessEndChildren which in turn calls endElementType it then calls pop() to pop the current state back to the "Schema" state.	endElementType finds that there was no actual child elements and that the content model is open, so it changes the default content model from EMPTY to ANY.
EndChildren "Schema"	ProcessEndChildren which finds that there is no cleanup function for the Schema then it pops the "Schema" state and we're back in the initial state.	
NotifyEvent ENDDOCUMENT	Finish	Checks for referenced but undeclared element types – and there are none, so we're done !